KARPOV.COURSES >>>
КОНСПЕКТ

> Конспект > 4 урок > Как спроектировать свою библиотеку факторов

> Оглавление

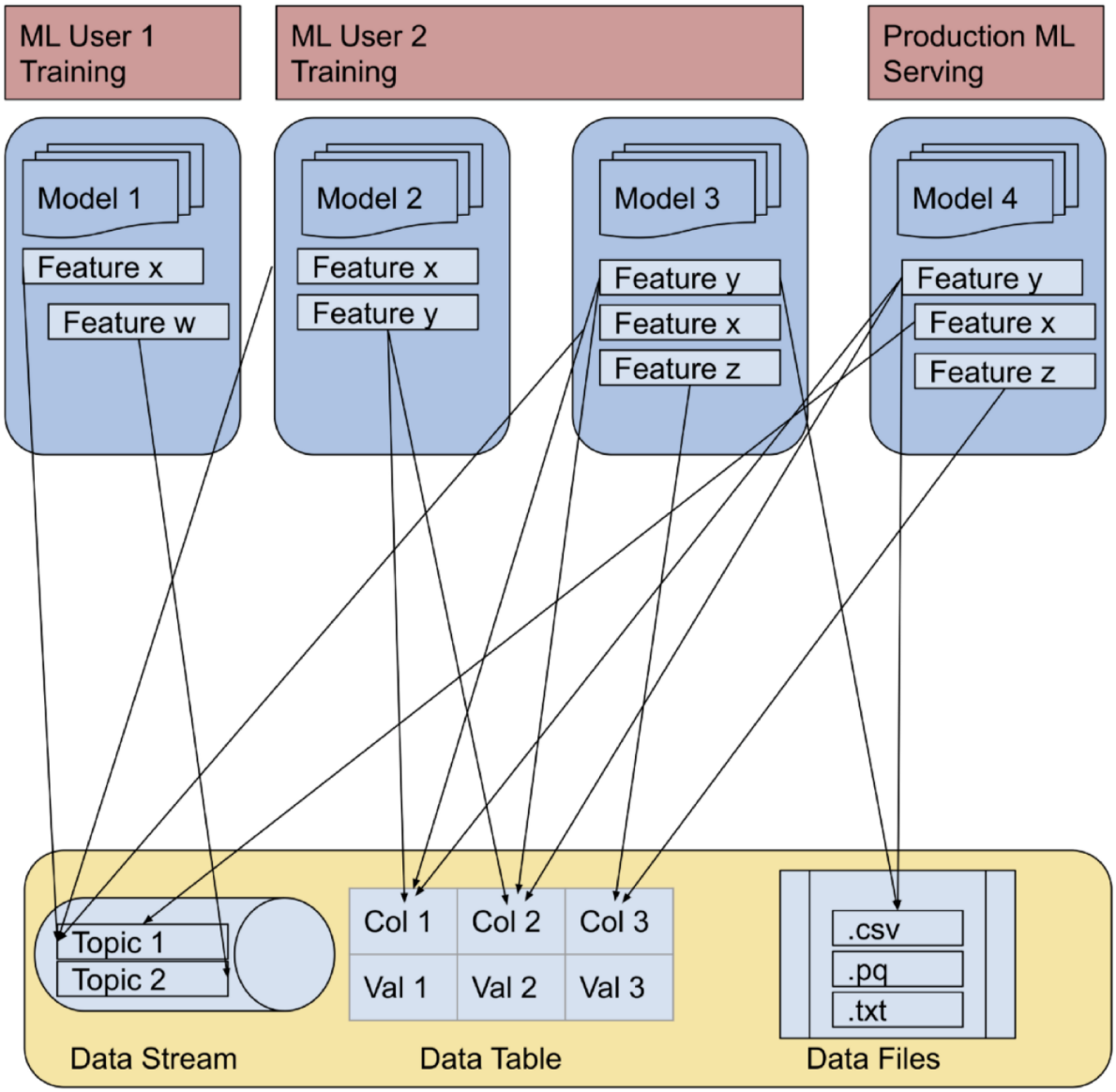
Для практики: модуль feature_impl и featurelib, ноутбук, данные из лекции 3.

> Что такое Feature Store

> Зачем нужно хранилище признаков

Рассмотрим жизненные ситуации, в которых чувствуется боль:

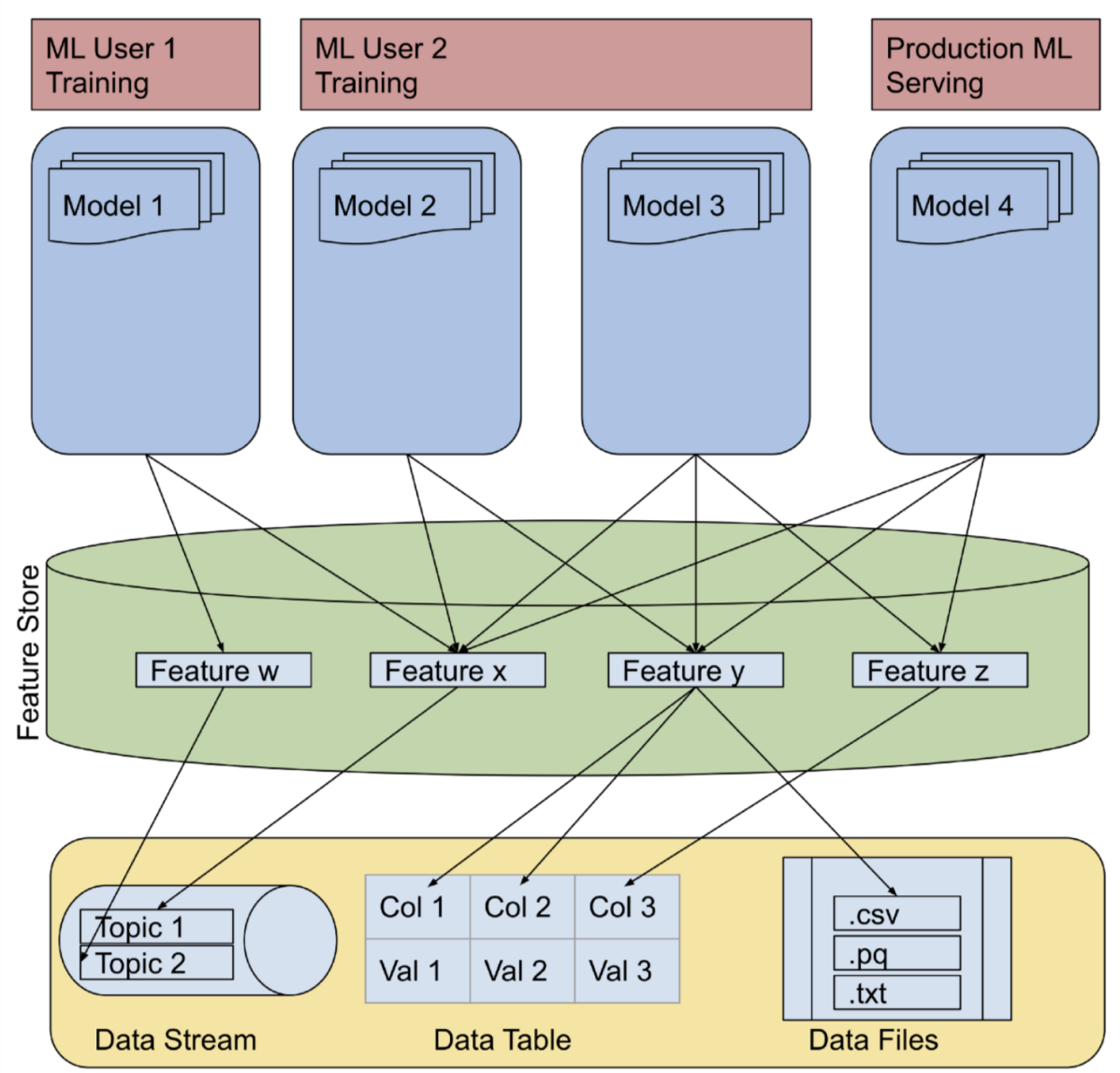
1. Вы потратили время, чтобы реализовать расчёт признака, который уже реализован коллегой (в вашей или “соседней” команде).
2. Коллега пришёл с вопросом: "Есть ли у нас расчёт такого фактора?"Увидел ваш код. Понял, что его трудно переиспользовать. Решил, что лучше напишет сам.
3. Скажем, есть DE/MLE. Вы как DS пишете расчёт фичей в неунифицированном формате.Для каждого отдельного проекта/приложения DE придётся разбираться с новым форматом и переписывать его.
4. Вы рассчитали признак, использовали его при обучении модели. Но он слишком долго считается, и поэтому его нельзя использовать в продакшене.



Каждый использует свои собственные фичи

> Что делать?

Организовать хранилище признаков (`Feature Store`).



Feature Store — промежуточный уровень между DS-ами и продакшеном с моделями, который позволяет не лезть в исходники данных

В Feature Store храним:

- Методологию расчёта признаков;
- Рассчитанные признаки (или вспомогательные данные для их быстрого расчёта).

> Плюсы Feature Store

- Нет расхождения значений признака у разных моделей, приложений, аналитиков.
- Reusability: нет дублирования работы аналитиков.

- Train/serve skew problem: и при прототипировании, и в продакшене признак считается по одной методологии.
- Discoverability: есть единое место, куда можно обратиться за признаками для модели, не нужно каждый раз лезть в разные источники данных и разбираться.
- Можно разбить работу: кто-то больше занимается разработкой и улучшением признаков, кто-то использованием их в моделях .
- Documentation: проще документировать расчёт признаков.

> Минусы Feature Store

- Высокая зависимость аналитиков и команд друг от друга. Может быть не удобно вносить необходимые отдельной команде изменения в методологию расчёта.

> Требования к Feature Store

Требования к FS и его дизайн сильно зависят от вашего бизнес-процесса.

Что важно:

- Насколько нужны свежие данные для расчёта?
- Как быстро протухает фактор?
- Сколько времени должен занимать расчёт?
- Часто или редко приходят запросы на расчёт факторов?
- Известно ли расписание расчётов?
- Сколько ресурсов(CPU, GPU, memory, disk) требуется для расчёта?
- Используется ли ML при расчёте данного фактора?
- Есть ли потребность в возможности считать фактор “в прошлом”?

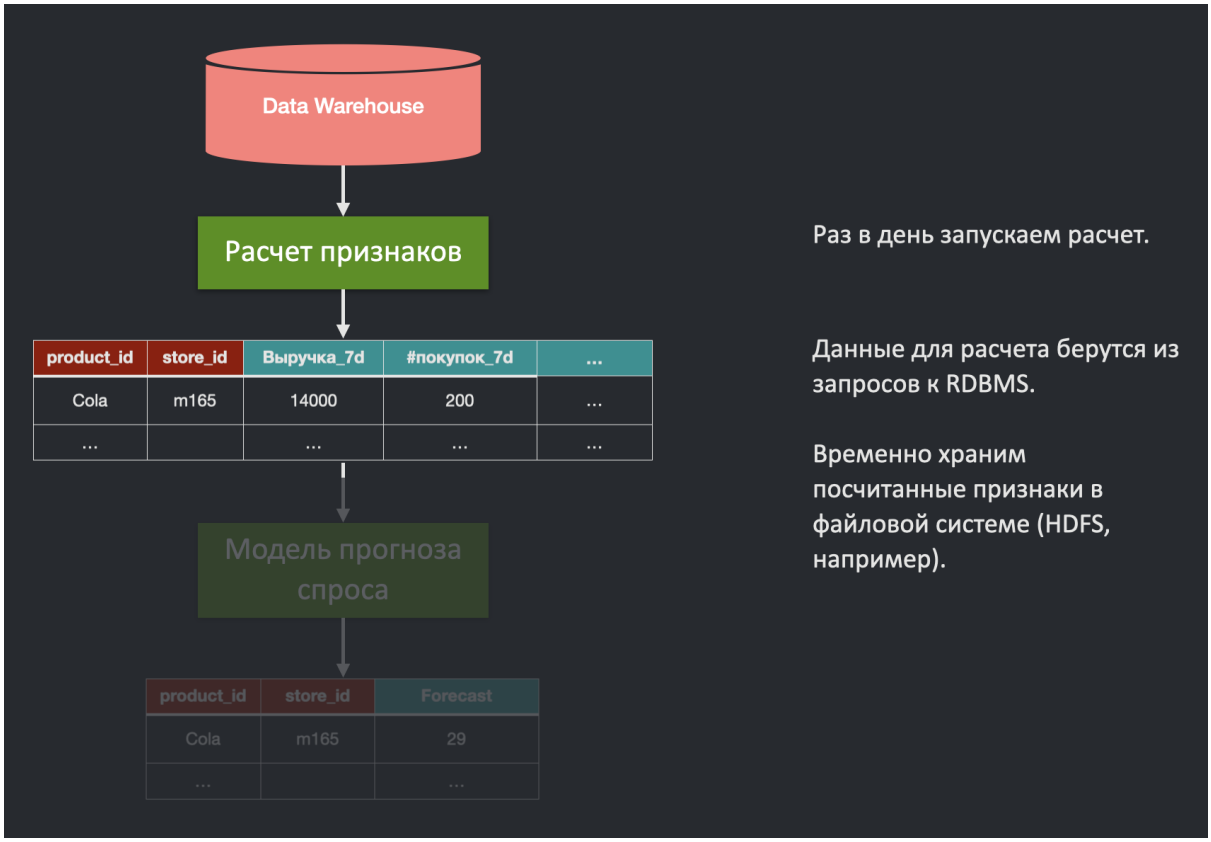
> Дизайн Feature Store. Простые примеры

> Прогноз спроса

Ситуация: нужно спрогнозировать, сколько товаров будет продаваться в конкретном магазине за период времени (три последующих дня).

Условия:

- Знаем расписание расчётов: нет ad-hoc запросов, запускаем ежедневно;
- Прогноз нужно делать по всему ассортименту;
- Время расчёта <= 8 часов.



Данные берутся напрямую из логов. Результат расчёта сохраняется в таблицу, которая используется для расчёта модели.

> Баннерная крутилка

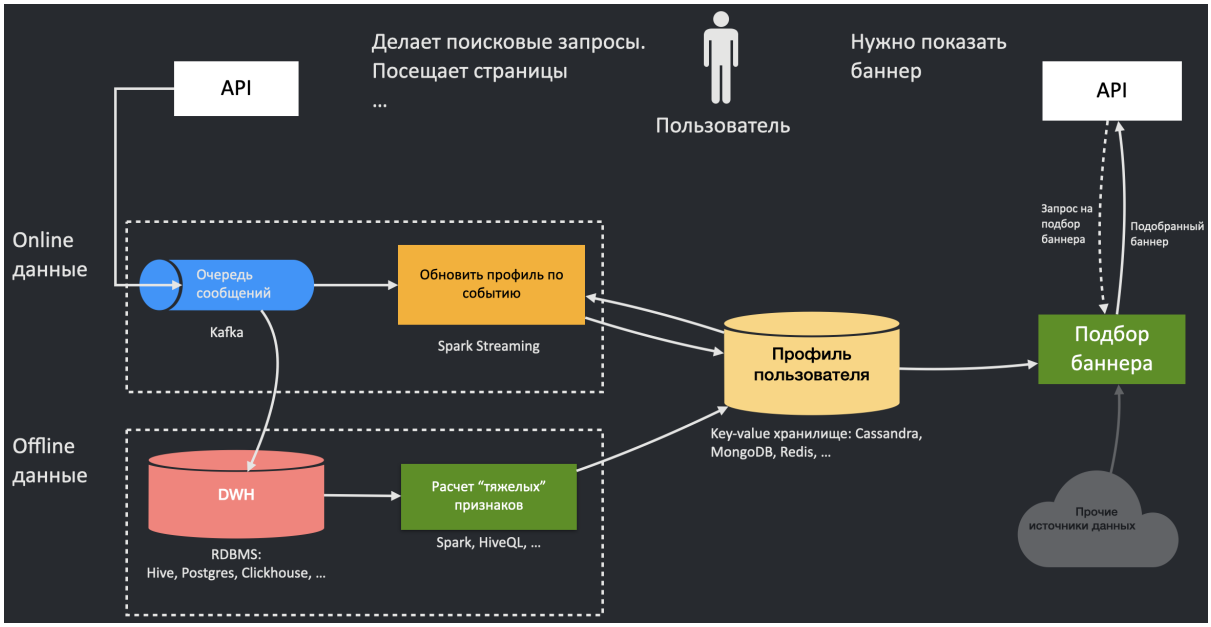
Ситуация: интернет-поисковик, у которого есть рекламная сеть с баннерами; есть данные о поисковых запросах клиентов, посещённых страницах и покупках в ваших сервисах.

На страницах есть место под баннер — необходимо решить, какой баннер показать.

Действия пользователя сразу же складываются в базу. На их основе сразу же обновляется профиль пользователя (лёгкие признаки), а "тяжёлые" признаки

обновляют профиль чуть позже при прохождении расчётов.

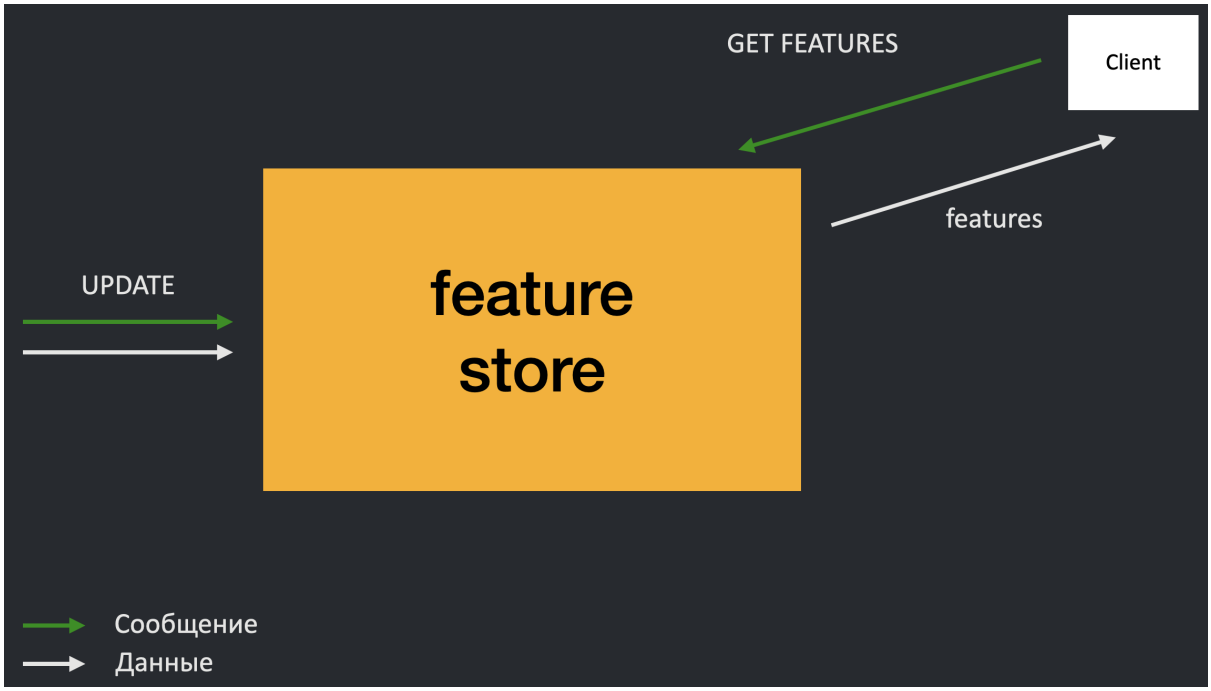
Пример лёгкого признака — последняя дата посещения магазина, а тяжёлого — средний чек. Дата просто обновляется, а чек необходимо рассчитывать.



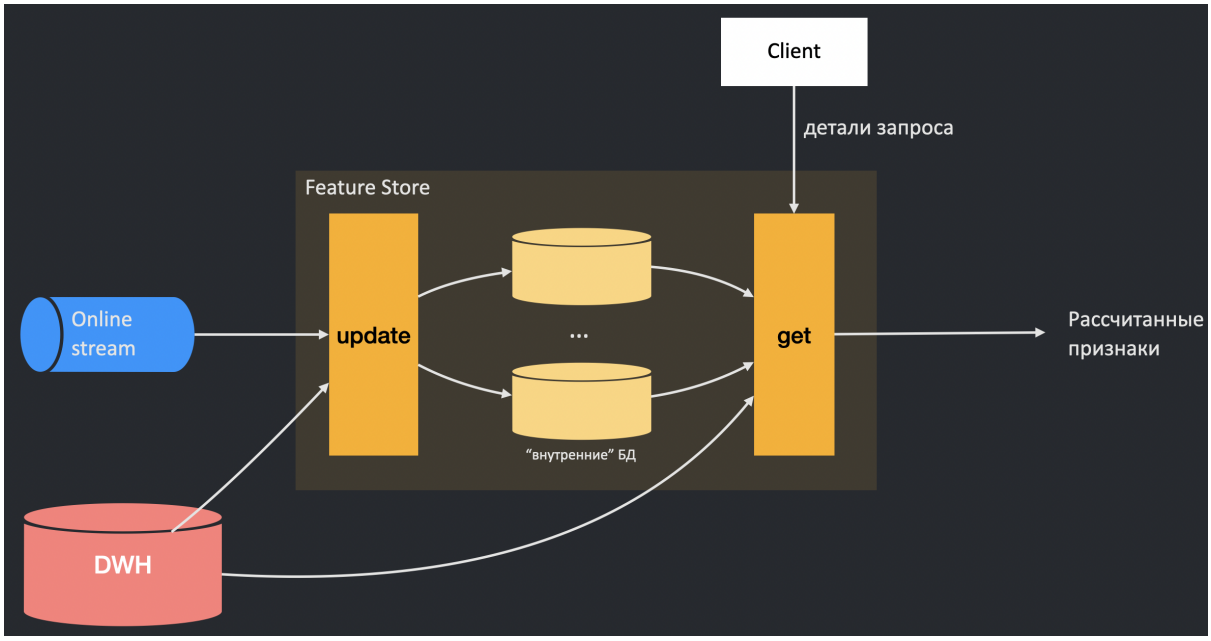
Система подбора баннера учтёт данные запроса, профиля и, возможно, иные источники. Получить ответ необходимо очень быстро.

Условия задач могут накладывать ограничения на используемые инструменты. В данном случае необходимо быстрое key-value хранилище.

> Дизайн Feature Store. Основные части



Хранилище — black box, где необходимо обновлять данные (менять состояние внутри хранилища) и откуда нужно извлекать данные

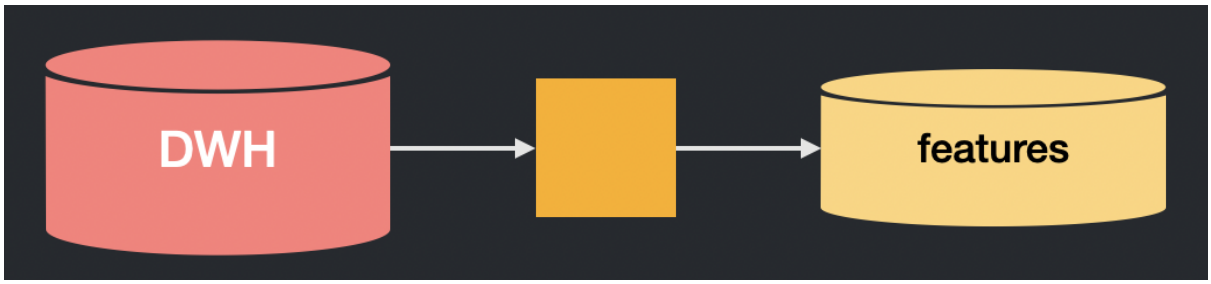


Заглянем внутрь хранилища: есть внутренние БД, где физически хранится информация, и два процесса по обновлению данных и получению данных

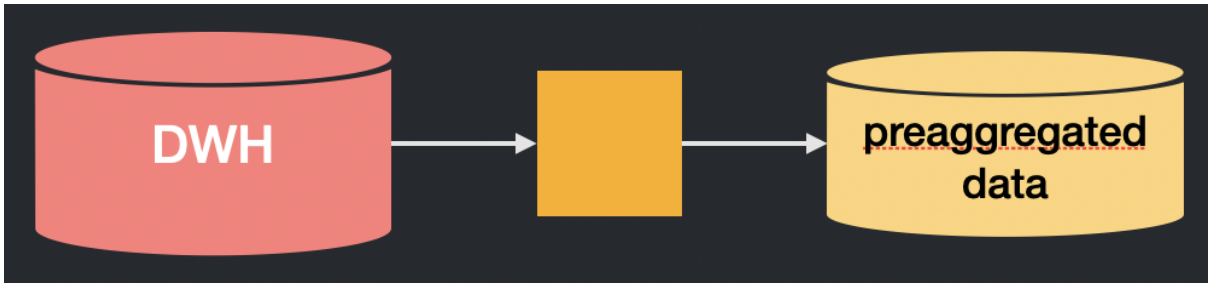
Нужно продумать:

- Библиотека факторов — методология расчёта признаков, "код";
- Обновление FS;
- Получение данных из FS.

> Приемы для обновления хранилища



1. Предварительный расчёт признаков



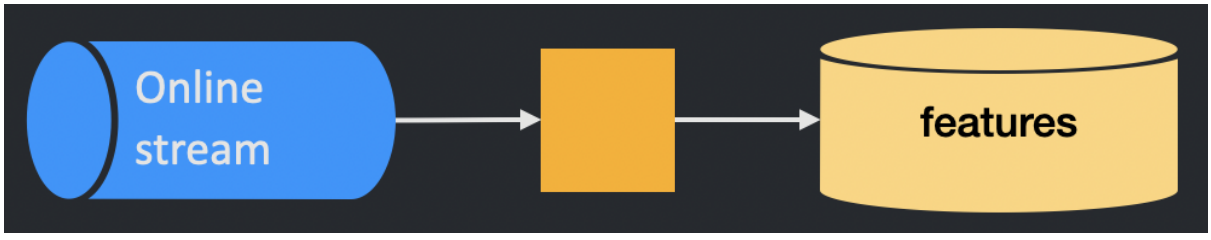
2. Предагрегация данных

Задача: по запросу (клиент, набор товаров) спрогнозировать, сколько рублей клиент потратит на набор товаров в следующем месяце.

Очевидно, важный признак — сколько клиент потратил на эти товары в предыдущем месяце.

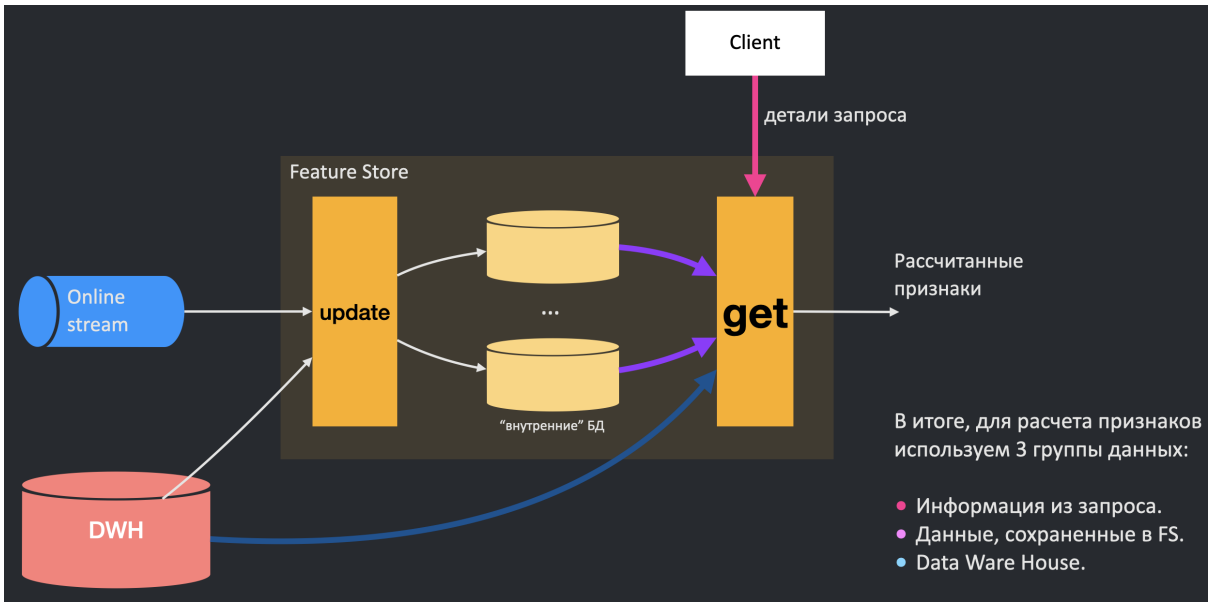
Но мы не можем хранить такой признак по всем возможным наборам категорий, поэтому:

- Храним предагрегированную таблицу (клиент, товар, выручка в прошлом месяце).
- По запросу считаем нужный признак.



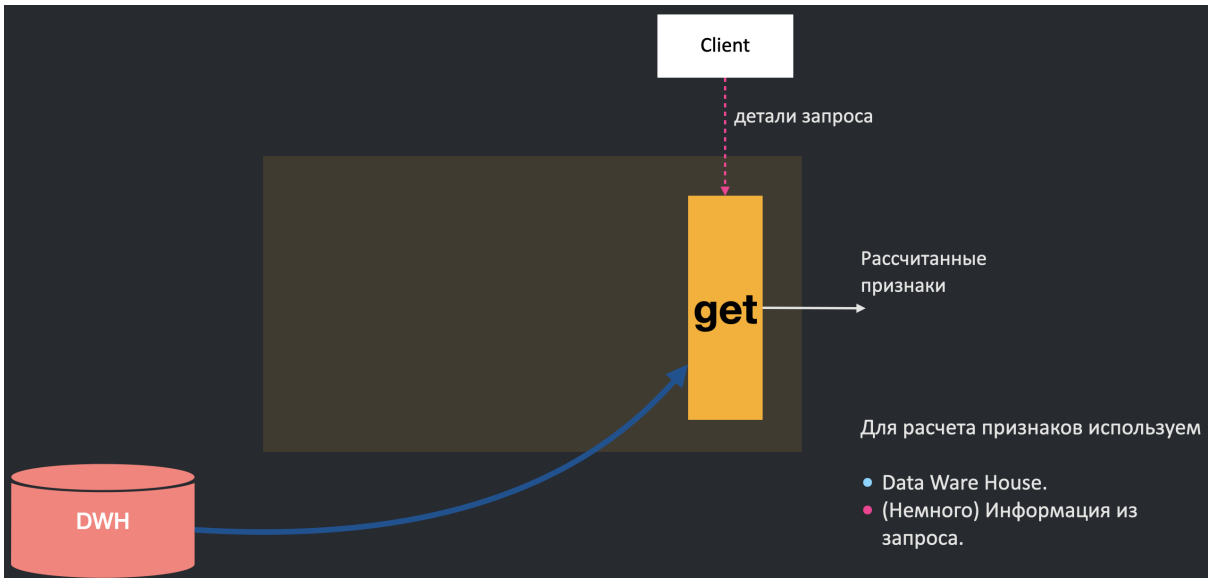
3. Обновление признаков по событиям

> Дизайн Feature Store. Получение данных



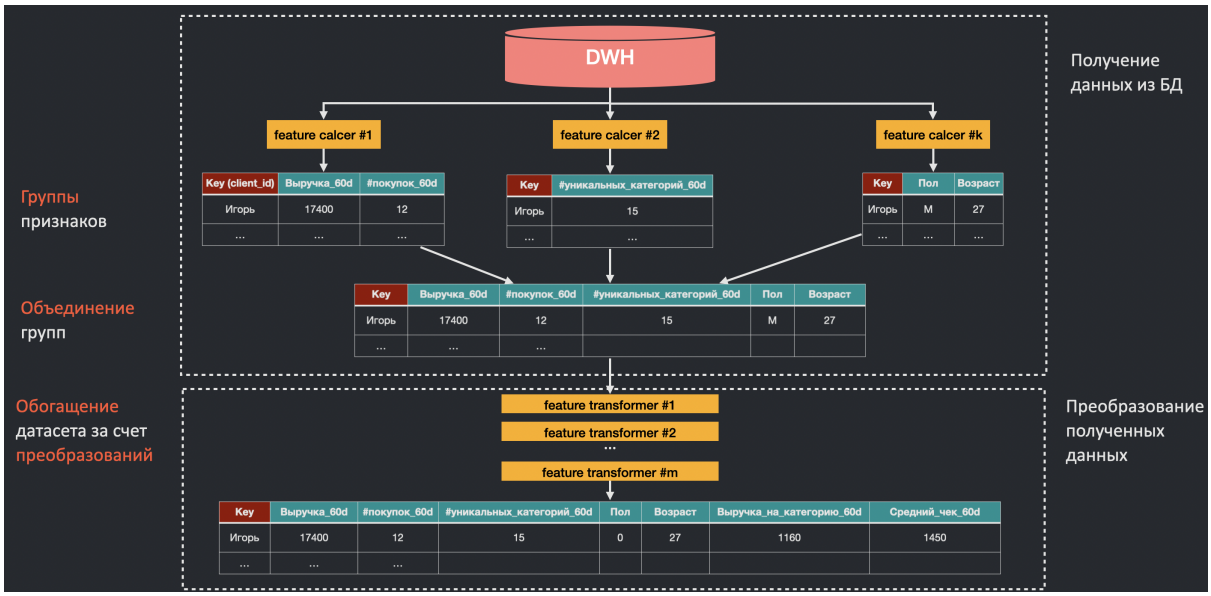
Для обновления данных в FS можно использовать поток событий и данные из DWH, а также информацию из запросов

Для начала рассмотрим упрощённый случай, когда физически данные не хранятся в хранилище (есть только библиотека для расчёта факторов). Вся используемая информация либо хранится в DWH, либо, возможно, мы получаем её из запроса.



Упрощённый случай

> Схема расчёта признаков



Процесс будет состоять их двух этапов:

1. Извлечение данных из DWH: отдельные скрипты для групп признаков.Это нужно, чтобы добавлять новые группы признаков и не менять расчёты старых. Разбивка на группы, чтобы считать сразу набор признаков, эффективнее с вычислительной точки зрения.
2. Преобразование данных.

> Создание библиотеки признаков. План-шпаргалка

1. Разобраться с инфраструктурой в вашей организации.
2. Оформить код для подключения к данным.
3. Задать шаблоны для расчёта признаков.
4. Для расчёта признаков писать код согласно шаблону.
5. Сделать автосборку документации признаков.
6. Сделать каталог признаков (UI).
7. Profit!

> Полезные ссылки

1. Michelangelo Palette: A Feature Engineering Platform at Uber
2. Zipline — Airbnb's Declarative Feature Engineering Framework
3. Цикл статей про Feature Stores
4. Хорошая книга по архитектуре
5. Библиотека для деплоя Feature Store